Grado en Ingeniería Informática

Trabajo Final de Grado

Reingeniería de procesos sobre un generador de *landings*

Realizado por:
Lara Gómez Carrera

Tutorizado por:
Mª José Aramburu Cabo

Fecha de lectura: 14 de Julio 2020

Resumen

Documentación del proceso de realización del proyecto *back-end* Do. Consiste en la reingeniería del proceso de creación de las páginas web dinámicas del sitio web DoYouSpain perteneciente a la empresa Gesmarket S.L.

Este proyecto se ha realizado durante una estancia en prácticas extracurriculares que ofrece la Universidad Jaume I y se ha presentado como trabajo final de grado de la asignatura EI1054, perteneciente al Grado en Ingeniería Informática.

El proyecto se ha desarrollado con las premisas y contexto establecidos por la empresa con el fin de poder sustituir el método de generación actual por el proyecto resultante.

Palabras clave

Landings dinámicas, Do, generador de páginas web dinámicas

Keywords

Dynamic landings, Do, Dynamic web page generator

Agradecimientos

La realización de este proyecto supone el final de una fase de formación muy importante en mi vida. Etapa de aprendizaje, sacrificio y dedicación que se ha hecho mas ameno con las personas con quién he compartido el camino. A estas personas es a quien quiero agradecer su tiempo y paciencia.

A mi familia, por su apoyo constante e incondicional.

A mis compañeros Yosra El Fakih, Carmen Moles, Paula Tomás, Paula González y Raúl Castellanos, por hacerlo todo mas ameno.

A mi tutora María José Aramburu, por sus exhaustivas correcciones y rapidez.

A la empresa *Gesmarket Internet para vender S.L.* por darme la confianza y oportunidad de continuar formándome con ellos. Y a mi tutor de las prácticas extracurriculares, David González, por guiarme en este mundo del sector servicios.

Índice general

1. Introducción **10**

1.1. Contexto y motivación del proyecto 10

1.2. Descripción del proyecto 11

1.3. Objetivos del proyecto 12

1.4. Estructura de la memoria 13

2. Planificación del proyecto **14**

2.1. Metodología 14

2.2. Planificación 14

2.3. Estimación de recursos y costes del proyecto 19

2.3.1. Recursos 19

2.4. Seguimiento del proyecto 22

3. Análisis y diseño del sistema **24**

3.1. Análisis del sistema 24

3.2. Especificación de requisitos 29

3.2.1. Requisitos de datos 29

3.2.2. Requisitos de tecnología 31

3.2.3. Requisitos de interfaces 32
3.2.4. Diseño de la arquitectura 33
3.2.5. Diseño de la interfaz 35

4. Implementación **40**

4.1. Encapsulamiento de datos 40
4.2. Modelo-Vista-Controlador 42
4.2.1. Modelo 42
4.2.2. Controlador 42
4.2.3. Vistas 44

5. Verificación y validación **46**

5.1. Test generar direcciones web 46
5.2. Test peticiones HTTP 47

6. Resultados y conclusiones **57**

6.1. Resultados 57
6.2. Conclusiones 58

Índice de figuras

2.1. Diagrama de Gantt 16

2.2. Diagrama de Gantt 18

2.3. Coste laboral hora efectiva T4 2019 22

3.1. Diagrama casos de uso 25

3.2. Resultados al aplicar SEM 26

3.3. Resultado al aplicar SEO 27

3.4. Obtención datos 30

3.5. Comparativa entre las formas de obtención de las vistas 33

3.6. Patrón diseño: Modelo-Vista-Controlador 34

3.7. Arquitectura de trabajo del servidor 35

3.8. Vista bloque *header* de la página web DoYouSpain 36

3.9. Vista bloque *form* tipo 1 de la página web DoYouSpain 36

3.10. Vista bloque *form* tipo 2 de la página web DoYouSpain 37

3.11. Vista bloque *form* tipo 3 de la página web DoYouSpain 37

3.12. Vista*form* tipo 4 de la página web DoYouSpain 38

3.13. Vista bloque *rents* de la página web DoYouSpain 38

3.14. Vista bloque *faqlist* de la página web DoYouSpain 39

3.15. Vista bloque *footer* de la página web DoYouSpain 39

4.1. Comunicacion de recursos a través del servicio web REST 41

4.2. Acceso a datos del objeto a través Viewbag 44

4.3. Uso marcado Razor 45

5.1. Petición HTTP para listado URLS 47

5.2. Petición HTTP para realizar peticiones 47

5.3. Resultado petición HTTP 48

5.4. Interfaz con el conjunto de resultados de varias peticiones 49

Índice de cuadros

2.1. Tabla amortizaciones bienes adquiridos. 22

3.1. Especificación caso de uso: Búsqueda de alquiler de coches. 28

3.2. Especificación caso de uso: Búsqueda de alquiler de coches. 29

3.3. Requisitos de datos para petición de una dirección web. 30

3.4. Requisitos de datos de consulta de una dirección web 31

5.1. Test evaluación aspectos generales 50

5.2. Test evaluación identidad e información 50

5.3. Test evaluación lenguaje y redacción 51

5.4. Test evaluación sobre rotulado 51

5.5. Test evaluación sobre *layout* 52

5.6. Test evaluación sobre estructura y navegación 53

5.7. Test evaluación realización búsquedas 54

5.8. Test evaluación elementos multimedia 54

5.9. Test evaluación ayuda ofrecida 54

5.10. Test evaluación accesibilidad 55

5.11. Test evaluación control y retroalimentación 56

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto que a continuación se detalla ha sido desarrollado en la empresa Gesmarket S.L., empresa dedicada a la prestación de servicios, tales como alquileres de vehículos de distintas compañías.
A través de un portal web, Gesmarket pone a disposición de los usuarios, vehículos de distintas compañías de alquiler. Su mercado está orientado a dar servicios en distintos países para cada uno de los cuales existe un *site* o sitio web distinto. Los principales *sites* son DoYouSpain[1] para el mercado de España, Carjet[2] orientado al mercado de Europa y DoYouItaly[3] para el mercado de Italia.
La empresa esta formada por alrededor de 50 empleados y se divide en varios departamentos. El departamento de atención al usuario, que atiende las dudas o problemas que puedan derivar de los usuarios de la web. El departamento administrativo, con fines administrativos y contables. El departamento de informática, que da asistencia a todos los problemas técnicos que puedan originarse en la web y la infraestructura que supone. El departamento de diseño, encargado de el diseño de las interfaces de usuario de todas las aplicaciones. Y por último, el departamento de marketing, encargado de dar y mejorar la visibilidad de las páginas web.
Las páginas web de los *sites* se crean a través de un generador de *landings*. Una *landing*

[1] www.doyouspain.com
[2] www.carjet.com
[3] www.doyouitaly.com

es una página web la cual es accedida tras pulsar en un enlace o botón situado en una guía, portal. La casuística de su generación, así como su complejidad, ha crecido tanto que ha surgido la necesidad de migrar el generador a una tecnología y arquitectura que aporte agilidad y flexibilidad.

1.2. Descripción del proyecto

El generador de *landings* actual, al cual haremos referencia por su nombre Generar Web, está desarrollado en ASP clásico [1] y Visual Basic, complementados con JavaScript, JQuery, Bootstrap, HTML5 y CSS3.
El ASP clásico se caracteriza por una estructura basada en ficheros. Estos ficheros contienen etiquetas HTML estático, etiquetas de *script* que se procesan en lado del servidor y etiquetas que se procesan en el lado del cliente. La construcción dinámica de las páginas web se va generando de forma lineal a través del documento. Esta fuerte vinculación entre la lógica de negocio y la interfaz de usuario, hace que la modificación o ampliación de alguna de sus partes, así como, del comportamiento del flujo de información que sucede entre ambas partes, sea una ardua tarea. Si a esto se añade el crecimiento de la casuística a tener en cuenta a la hora de generar, así como, la ampliación y mejora constante de la interfaz de usuario, la complejidad de hacer dichas tareas es aún mayor.

Estas condiciones han dado lugar a la necesidad de migrar el generador a una arquitectura de software y tecnología que aporte flexibilidad, escalabilidad y facilidad de mantenimiento. De esta forma nace Do, un nuevo generador que tiene que replicar el funcionamiento de Generar web. La diferencia entre ambos reside en su diseño, su funcionamiento y lenguaje. Ahora se aplicará una arquitectura MVC, Modelo-Vista-Controlador [2], de esta forma estarán totalmente desacopladas la parte lógica de la del diseño y del acceso a los datos. Todas las peticiones a la base de datos se encapsulan a través de una API [3], también conocido como conjunto de subrutinas, y el lenguaje utilizado para la programación de todo el proyecto será C#. Esto dará lugar a un generador mucho más dinámico, en el que las operaciones de control de errores, ampliación y modificación serán mucho más sencillas.

Este proyecto ha consistido en la migración del generador de *landings* dinámicas a este nuevo generador.

1.3. Objetivos del proyecto

El objetivo principal del proyecto es la replicación del comportamiento de Generar Web para las *landings* dinámicas en la nueva arquitectura y tecnología elegidas. Este objetivo es tan amplio que para una mejor compresión y desarrollo se subdivide en objetivos menores.

Análisis

Analizar el comportamiento lógico de cómo deben comportarse las *landings*, qué información forma cada una de ellas y las propiedades que condicionan la información a imprimir.

Definición arquitectura

Desarrollo de la arquitectura MVC, Modelo-Vista-Controlador, con las siguientes tareas:

- Análisis y estructuración del contenido de cada capa.
- Definición de los modelos.
- Definición de los controladores.
- Definición de vistas.
- Verificación y validación de las definiciones.
- Implementación de los modelos, controladores y vistas.

Validación y verificación

Validación de requisitos, control de errores y testeo del correcto funcionamiento de la arquitectura implementada.

Todos estos objetivos deben hacer cumplir el objetivo organizativo de mejorar la forma de generación para que sea más sencillo realizar cambios y mejoras sobre las *landings*. Esto hará más atractiva la experiencia del usuario, repercutiendo a su vez en un aumento de captación de nuevos usuarios.

1.4. Estructura de la memoria

La memoria se estructurará en varios capítulos:

Capítulo 2, descripción de la planificación, metodología, recursos y costes llevada a cabo durante el desarrollo del proyecto.

Capítulo 3, análisis llevado a cabo previamente para el conocimiento tanto del contexto del programa como del funcionamiento deseado. Este análisis dará como resultado una serie de requisitos que serán usados para el posterior diseño del programa.

Capítulo 4, aspectos más relevantes de la implementación del programa.

Capítulo 5, test creados para realizar la verificación y validación del desarrollo, teniendo en cuenta distintos criterios y métodos.

Capítulo 6, conclusiones resultantes tras la realización del programa, así como de futuras mejoras que se podrían aplicar.

Capítulo 2

Planificación del proyecto

2.1. Metodología

La metodología seguida ha sido la metodología ágil Scrum [4], se ha seguido esta metodología ya que es la que se aplica en la empresa.

Semanalmente se realizan reuniones de todos los subgrupos que componen el departamento de informática: grupo de desarrollo *backend*, grupo de desarrollo *front-ed*, departamento de diseño y marketing. En esas reuniones se establecen los objetivos conseguidos y se definen nuevos a la par que se plantean discrepancias, cambios y los problemas surgidos durante el desarrollo.

Esta metodología se ha visto beneficiada con el uso de herramientas software tales como Git para compartir el desarrollo y control de versiones. Trello como herramienta organizativa de trabajo, así como herramientas sociales de comunicación como Hangouts.

2.2. Planificación

Para tener una organización de tareas y así poder estimar el alcance temporal del proyecto, inicialmente se han dividido los objetivos en objetivos mas específicos y clasificados según su funcionalidad. De esta forma a la par que sirve como guía a seguir en el desarrollo, permite conocer el estado del proyecto en un momento concreto.

ID	Tipo	Actividad	Tiempo	Antecesora
		Do	43 días	
1		**Análisis**	**2,88 d**	
1.1	Tarea	Estudio y definición de requisitos	9,6 h	
1.2	Tarea	Definición alcance y objetivos	3,57 h	
1.3	Tarea	Definición diagramas casos de uso	7 h	
2		**Planificación**	**2 d**	1
2.1	Tarea	Definición de las actividades	5 h	1.3
2.2	Tarea	Estimación temporal y de recursos	5 h	2.1
2.3	Tarea	Verificación y validación	4 h	2.1
3		**Diseño**	**3 d**	
3.1	Tarea	Diseño funcional	14h	
3.2	Tarea	Arquitectura	7 h	3.1
4		**Implementación**	**27,6 d**	
4.1	Tarea	Formación con entorno	56,6 h	
4.2	Tarea	Programación	136 h	3
5	Hito	**Pruebas**	**7.57 d**	
5.1	Tarea	Pruebas funcionamiento	24h	
5.2	Tarea	Pruebas unitarias	15 h	
5.3	Tarea	Pruebas de integración	14 h	5.1;5.2
6	Hito	**Preparación documentación técnica**		
6.1	Tarea	Documentación propuesta técnica	17 h	5
6.2	Tarea	Documentación memoria TFG	120 h	6.1
6.3	Tarea	Preparación presentación TFG	13 h	6.1;6.2
6.4	Hito	Defensa TFG		

A continuación, para un mayor detalle del coste temporal de cada tarea, se adjunta el diagrama de Gantt y un diagrama de descomposición de tareas. La inversión semanal en la empresa era de 35 horas, 7 horas diarias.

	Task	Assigned To	Start	End	Dur
	Do	450 h	17/10/19	1/6/20	163
1	**Análisis**		17/10/19	21/10/19	3
1.1	Definición de las actividades	9.6 h	17/10/19	18/10/19	2
1.2	Estimación temporal y de recursos	3,57 h	18/10/19	18/10/19	1
1.3	Verificación y validación	4 h	21/10/19	21/10/19	1
2	**Planificación**		22/10/19	25/10/19	4
2.1	Definición de las actividades	5 h	22/10/19	24/10/19	3
2.2	Estimación temporal y de recursos	5 h	25/10/19	25/10/19	1
2.3	Verificación y validación	4 h	25/10/19	25/10/19	1
3	**Diseño**		24/10/19	29/10/19	4
3.1	Diseño funcional	14 h	24/10/19	25/10/19	2
3.2	Arquitectura	7 h	26/10/19	29/10/19	2
4	**Implementación**		29/10/19	29/11/19	22
4.1	Formación con entorno	56,6 h	29/10/19	8/11/19	9
4.2	Programación	136 h	30/10/19	29/11/19	21
5	**Pruebas**		9/12/19	21/1/20	31
5.1	Pruebas funcionamiento	24 h	9/12/19	15/1/20	27
5.2	Pruebas unitarias	15 h	16/1/20	17/1/20	2
5.3	Pruebas de integración	14 h	18/1/20	21/1/20	2
6	**Preparación documentación técnica**		22/1/20	1/6/20	94
6.1	Documentación propuesta técnica	17 h	22/1/20	1/3/20	28
6.2	Documentación memoria TFG	120 h	2/3/20	1/5/20	45
6.3	Preparación presentación TFG	13 h	2/5/20	8/5/20	5
6.4	Defensa TFG	0 h			

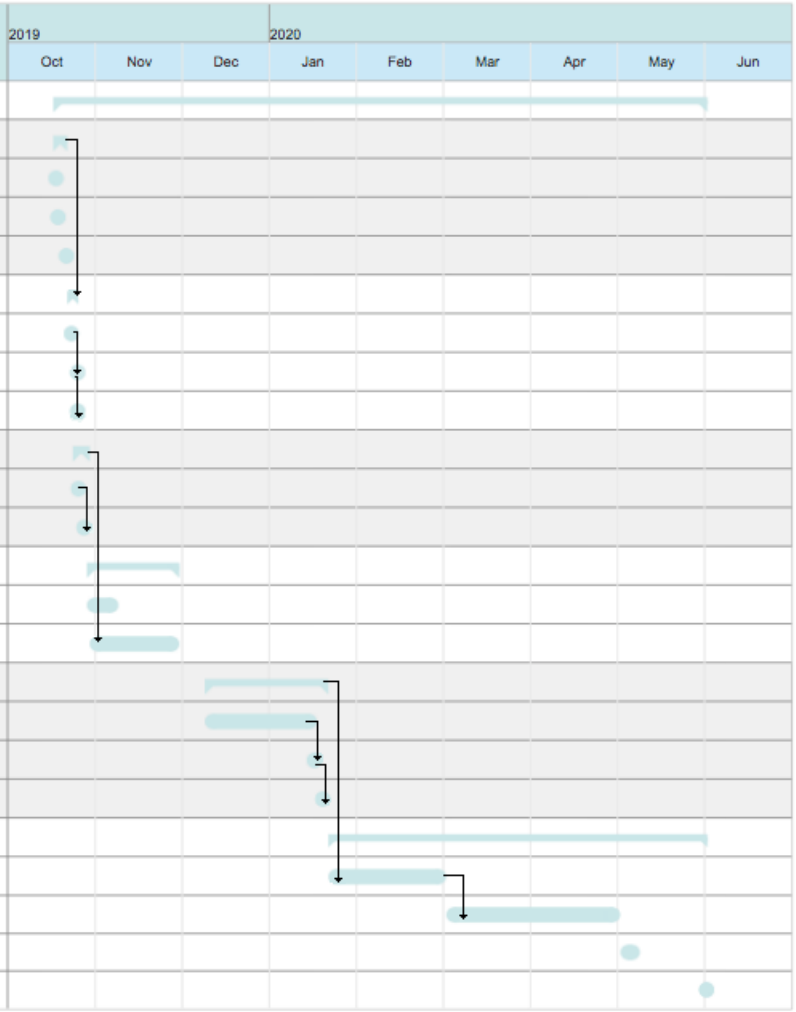


Figura 2.1: Diagrama de Gantt

Esta planificación inicial finalmente no fue la seguida, la proclamación del estado de alarma en el mes de marzo por la pandemia del Covid-19 con el consiguiente confinamiento supuso un retraso en los procesos de documentación de la memoria. A continuación, en la figura 2.2 se muestra el diagrama de Gantt con la desviación. La distribución de tareas es la misma, pero varía el número de días invertido, siendo superior en la planificación final.

	Task	Assigned To	Start	End	Dur
	Do	450 h	17/10/19	1/7/20	182
1	**Análisis**		17/10/19	21/10/19	3
1.1	Estudio y definición de requisitos	9.6 h	17/10/19	18/10/19	2
1.2	Definición alcance y objetivos	3,57 h	18/10/19	18/10/19	1
1.3	Definición diagramas casos de uso	7 h	21/10/19	21/10/19	1
2	**Planificación**		22/10/19	25/10/19	4
2.1	Definición de las actividades	5 h	22/10/19	24/10/19	3
2.2	Estimación temporal y de recursos	5 h	25/10/19	25/10/19	1
2.3	Verificación y validación	4 h	25/10/19	25/10/19	1
3	**Diseño**		24/10/19	29/10/19	4
3.1	Diseño funcional	14 h	24/10/19	25/10/19	2
3.2	Arquitectura	7 h	26/10/19	29/10/19	2
4	**Implementación**		29/10/19	29/11/19	22
4.1	Formación con entorno	56,6 h	29/10/19	8/11/19	9
4.2	Programación	136 h	30/10/19	29/11/19	21
5	**Pruebas**		9/12/19	21/1/20	31
5.1	Pruebas funcionamiento	24 h	9/12/19	15/1/20	27
5.2	Pruebas unitarias	15 h	16/1/20	17/1/20	2
5.3	Pruebas de integración	14 h	18/1/20	21/1/20	2
6	**Preparación documentación técnica**		22/1/20	1/7/20	116
6.1	Documentación propuesta técnica	17 h	22/1/20	1/3/20	28
6.2	Documentación memoria TFG	120 h	2/3/20	29/6/20	55
6.3	Preparación presentación TFG	13 h	16/5/20	29/6/20	5
6.4	Defensa TFG	0 h			

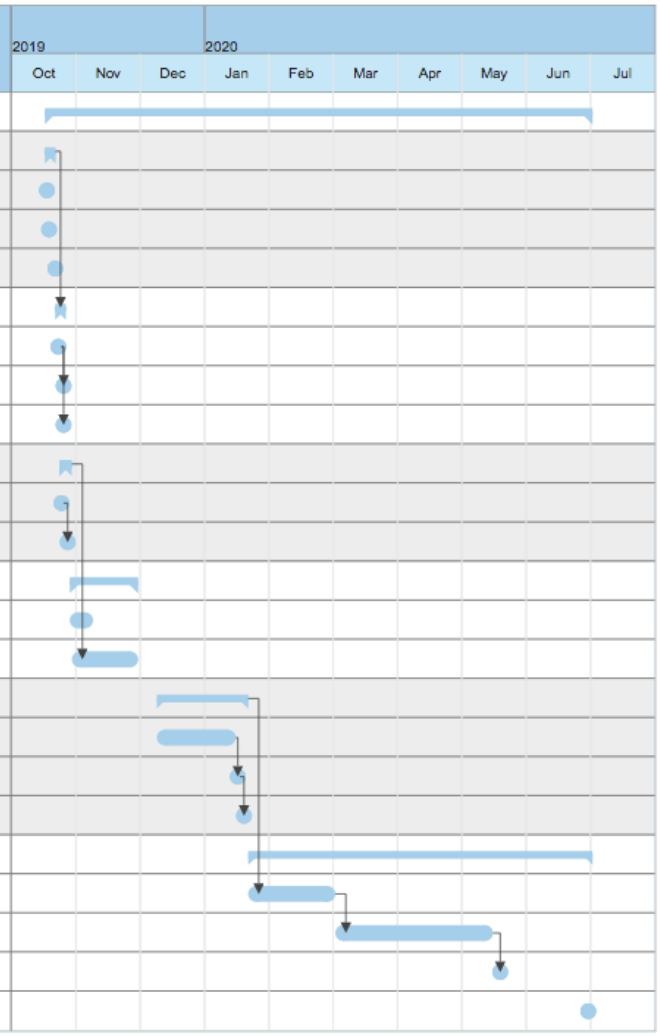


Figura 2.2: Diagrama de Gantt

2.3. Estimación de recursos y costes del proyecto

2.3.1. Recursos

Los recursos que han sido necesarios para el desarrollo del proyecto han consistido como base un ordenador con conexión a Internet. También han sido necesarias un conjunto de herramientas tecnológicas para poder desarrollar y emular lo desarrollado.

Herramientas tecnológicas

Visual Studio 2019
: Herramienta software privativa de Microsoft. Ofrece un entorno de desarrollo integrado, pudiendo crear aplicaciones en cualquier entorno que soporte la plataforma .NET [5]. Esta ha sido la herramienta principal de desarrollo, en ella se ha especificado toda la estructura y comportamiento de la aplicación.

Git
: Herramienta de software libre de control de versiones [6]. Se ha requerido de esta herramienta para una óptima gestión de los cambios del código desarrollado.

TortoiseGit
: Herramienta de software libre. Entorno gráfico que facilita la interoperabilidad con *Git*. Esta aplicación genera un repositorio local, sobre la cual el usuario realiza todas las operaciones de control de versiones requeridas [7]. Es la propia aplicación quien se encarga de gestionar la sincronización de todas las operaciones con el repositorio de *Git*. Esta herramienta es la que se ha utilizado de forma diaria para realizar todas las operaciones de control de versiones.

SQL Server Management Studio
: Herramienta software privativa de Microsoft. Entorno integrado para la gestión integral de bases de datos relacionales [8]. Esta herramienta ha sido utilizada para obtener y consultar todos los datos que se almacenan en la base de datos. Estos datos daban soporte a los textos utilizados en las *landings* así como parámetros necesarios para la creación de direcciones web válidas.

Internet Information Services
: Herramienta software privativa de Microsoft. Entorno que presta el servicio de emulación de un servidor web, de tal manera que se pueda publicar páginas web local-

mente [9]. Esta herramienta ha sido utilizada para controlar y gestionar el correcto funcionamiento de construcción de las *landings*.

Visor de Eventos

Herramienta software privativo de Microsoft. Aplicación que administra la información que producen toda actividad ocurrida en el ordenador [10]. Herramienta necesaria cuando se ha producido un error y, ni el gestor de excepciones de Microsoft Visual Studio, ni el depurador del ISS ofrece información relevante para conocer el origen del error.

Postman

Herramienta de software que permite testear una API sin la necesidad de instalar un cliente [11]. Inicialmente se creó como una extensión de Google Chrome [12], pero actualmente también existe una aplicación de escritorio.

Google Chrome

Herramienta software privativa de Google. Navegador web utilizado para visualizar las páginas web. A parte de la visualización ofrece unas herramientas de inspección del código fuente de la página web que han sido muy útiles para el control de errores.

Lenguajes de programación

C#

Lenguaje de programación descriptivo de Microsoft [13]. Utilizado para definir y describir la arquitectura y lógica del generador. Un ejemplo de uso es la comunicación entre capas.

JavaScript

Lenguaje de programación interpretado [14]. Utilizado para desarrollar la parte interactiva de las vistas, por ejemplo la selección del destino o fechas en el formulario de búsqueda.

HTML5

Lenguaje de marcado utilizado para elaborar páginas web [15]. Utilizado para la elaboración de las vistas.

CSS (Cascading Style Sheets)

Lenguaje utilizado para definir el diseño de las vistas [16].

SQL (Structured Query Language)

Lenguaje de consulta para datos estructurados [17]. Lenguaje diseñado para administrar y recuperar información de sistemas de gestión de bases de datos relacionales. En este proyecto ha sido utilizado para almacenar toda la información que muestra cada *landing* así como información necesaria para su construcción.

ASP.Net (Active server Pages)

Entorno web distribuido por Windows que permite la creación de aplicaciones web. Este es el entorno en el cual está escrito el actual generador, por lo que saber interpretar su lenguaje ha sido necesario para entender el comportamiento del generador.

Visual Basic

Lenguaje de programación dirigido por eventos [18]. Lenguaje utilizado para definir el comportamiento del actual generador, ha sido necesario su aprendizaje para entender su funcionamiento.

La elección de las tecnologías anteriormente descritas, está condicionada principalmente por la arquitectura del servidor donde se ejecuta la aplicación. Do no trabaja de forma aislada, interactúa con otras aplicaciones, todas deben estar implementadas en el mismo *framework*. Las herramientas y tecnologías complementarias utilizadas para la realización de los test fueron seleccionadas por consejos de los compañeros/as.

Costes

Para el desarrollo del proyecto se tuvo que realizar una inversión en bienes materiales para adquirir el mobiliario, máquina y periféricos necesarios. El mobiliario ha consistido en un mesa y una silla, la máquina un ordenador y los periféricos dos monitores, un soporte para los monitores y un ratón. No hubo necesidad de adquirir ninguna licencia de software específica, las licencias concedidas formaban parte de un paquete de licencias adquirido con anterioridad a mi incorporación cuyo coste prácticamente ya ha sido amortizado. El coste de adquisición de cada bien se detalla a continuación.

Bien	Unidades	P.V.P	Amortización	P.V.P amortizado
Escritorio	1	100€	10 %	2,5€
Silla oficina	1	60€	10 %	1,5€
Ordenador	1	610,54€	25 %	38,15€
Monitor	2	109€	20 %	10,90€
Soporte	1	31,78€	10 %	0,79€
Ratón	1	4,25€	20 %	0,21€
Importe total:	7	915,57	-	54,05€

Cuadro 2.1: Tabla amortizaciones bienes adquiridos.

El calculo del importe de amortización de los bienes se ha obtenido según lo que establecen las tablas de amortización del Impuesto sobre Sociedades [19]. A estos costes también hay que añadir el coste laboral por hora efectiva que establece el Instituto Nacional de Estadística.

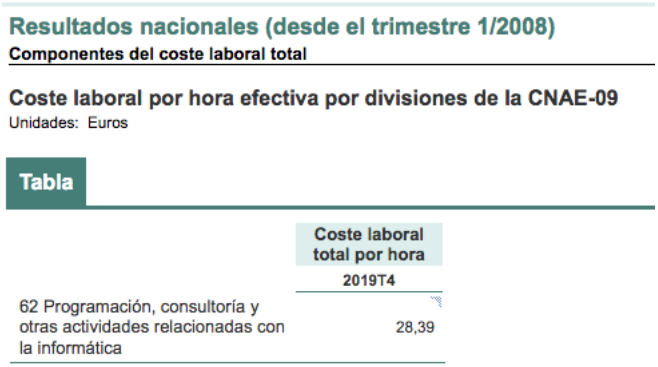


Figura 2.3: Coste laboral hora efectiva T4 2019

Este coste multiplicado por las 300 horas de duración del desarrollo equivale a 8.517€, si ha esto se le suma los gastos anteriormente calculados, el coste total del proyecto asciende a 8.571,05€.

2.4. Seguimiento del proyecto

El seguimiento del proyecto se ha realizado mediante las reuniones semanales llevadas a cabo por los informáticos especialistas en *front-end*, así también con los especialistas en *back-end*.

En las reuniones semanales con los especialistas en *fron-end* se establecían los cambios que debían propagarse a todas las *landings* en base a los resultados obtenidos de los experimentos realizados sobre la página web. Todos estos cambios debían aplicarse a las *landings* en producción así como, las que se generasen con Do.

En las reuniones semanales con los especialistas en *back-end* se exponían los progresos así como modificaciones o mejoras a aplicar en lo desarrollado hasta el momento. Las reuniones fueron ganando en contenido a la par que el proyecto iba avanzando. Aunque las modificaciones han sido constantes, han servido para aprender buenas prácticas en el desarrollo de software.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

Con el fin de replicar el comportamiento integro de forma exitosa del actual generador ha sido importante analizar previamente su funcionamiento y características. Como resultado de este análisis se ha creado un diagrama de casos de uso. Este tipo de diagrama especifica la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. Si bien el diagrama abarca toda la funcionalidad llevada a cabo por el generador, el alcance de este proyecto se limita al caso de uso de búsqueda alquiler vehículos. Tal y como muestra la figura 3.1, la actividad se inicia cuando un usuario esta interesado en el alquiler de un coche, cuando realiza dicha búsqueda se muestra un conjunto de enlaces los cuales dan acceso a diferentes páginas web que pueden ofrecer alquileres.

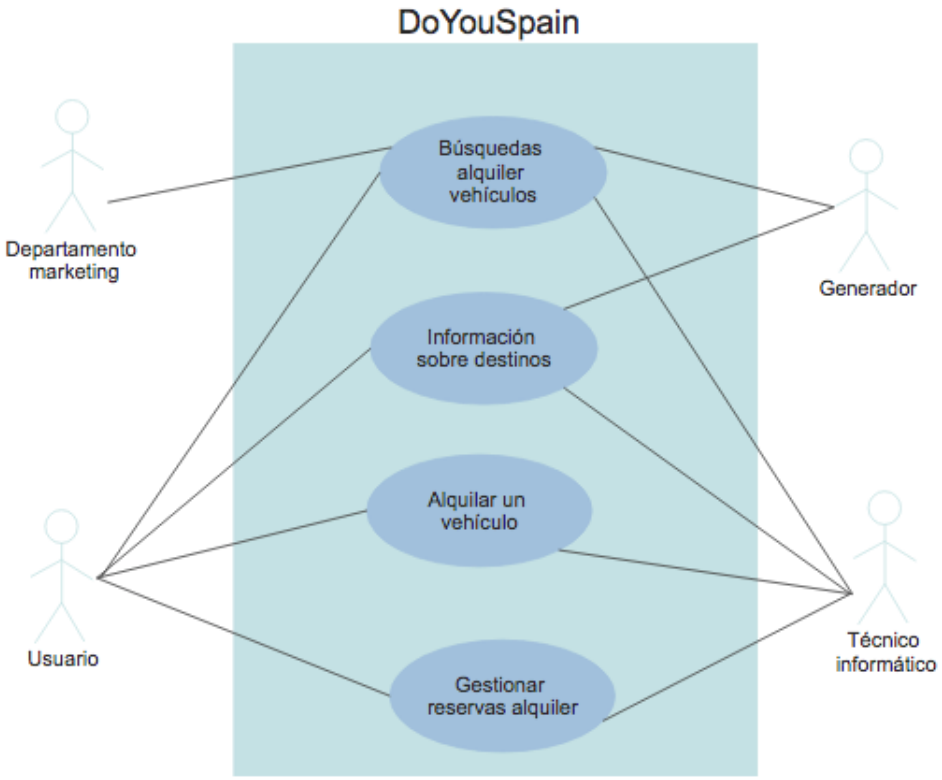


Figura 3.1: Diagrama casos de uso

El modelo de negocio establece que el usuario acceda a cualquiera de los enlaces que hagan referencia a la empresa. El acceso exitoso del usuario a estos enlaces se puede conseguir mediante la aplicación de políticas de posicionamiento en los motores de búsqueda.

El departamento de marketing hace uso de estas políticas y como herramienta de optimización necesitan del uso de *landings* dinámicas. Antes de explicar el por qué de la necesidad de construir este tipo de *landings*, se explicará a continuación qué son las políticas de posicionamiento.

En el listado de enlaces que se muestra tras realizar un búsqueda, su orden de impresión así como su estructura está regulado por las políticas de posicionamiento de cada motor de búsqueda. Estas políticas establecen un conjunto de reglas y estándares en la definición y diseño de las páginas web cuyo cumplimiento en su diseño y programación se ve retribuido en una mejor posición respecto otras que o bien no lo cumplan o lo cumplan peor. Existen dos tipos de posicionamiento, el posicionamiento natural también denominado SEO y el posicionamiento SEM. La diferencia entre ambos radica en que si bien el SEO depende de las propias características que definen la web y su nivel de cumplimiento

de los estándares, el SEM busca la visibilidad mediante campañas de anuncios de pago.

Tal y como muestra la figura 3.2, los primeros resultados en mostrarse tras la búsqueda son los pertenecientes a las campañas de SEM, aquella empresa que personaliza más el resultado acorde a la búsqueda realizada tendrá mas opciones de ser visitada.

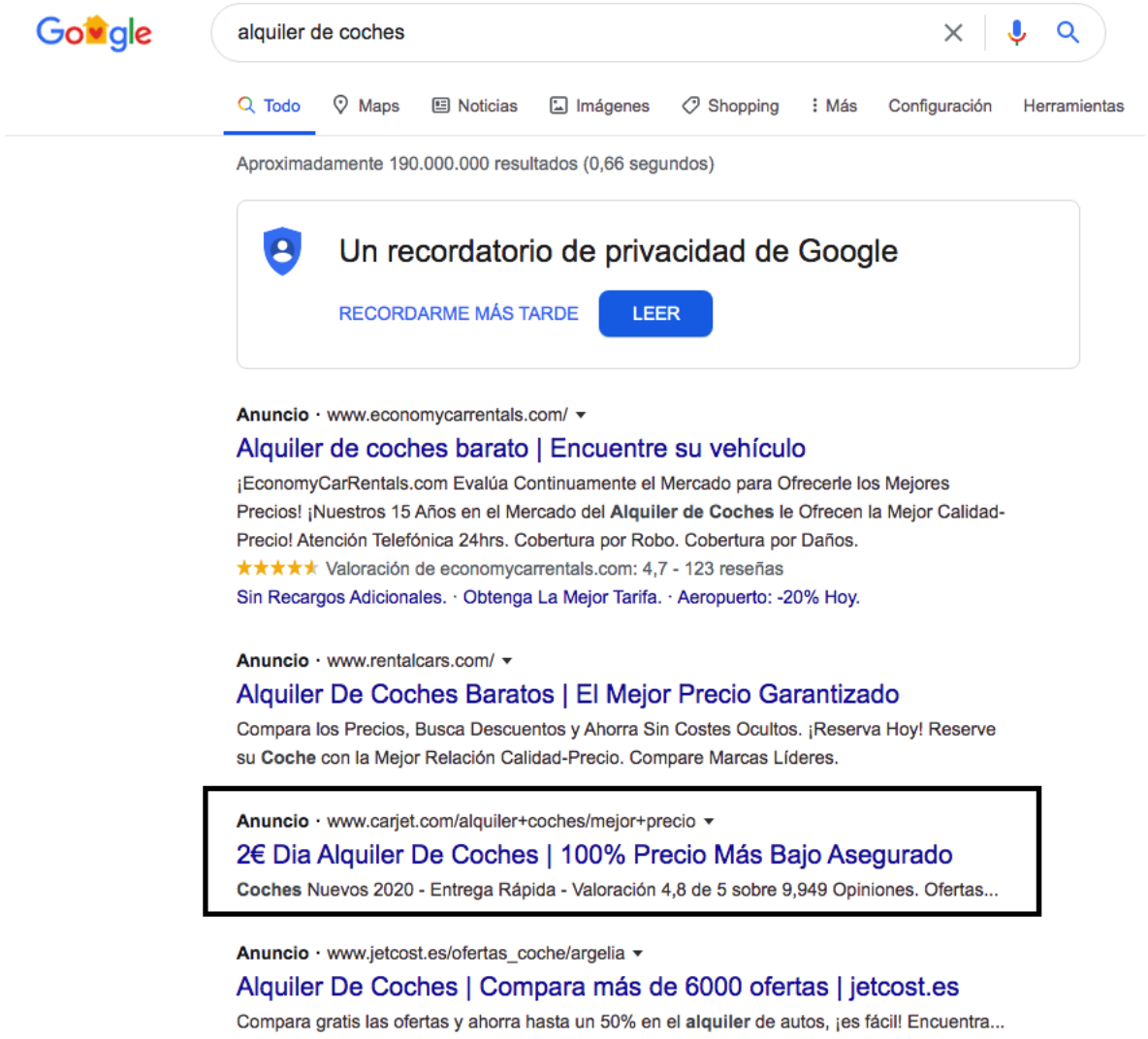


Figura 3.2: Resultados al aplicar SEM

Tras los resultados de SEM, se listan los resultados de SEO (figura 3.3), también denominados orgánicos ya que su posicionamiento sólo depende de su definición.

Figura 3.3: Resultado al aplicar SEO

El departamento de marketing encargado de gestionar la parte de SEM, utiliza las *landings* dinámicas para personalizar los anuncios ajustándolos a una serie de parámetros que se dan cuando se realizan búsquedas tales como palabras claves. Esto da lugar a la necesidad de tener una serie de parámetros que puedan ir variando en las direcciones web del sitio web y que den lugar a *landings* personalizadas acorde a los parámetros introducidos en la URL. Indicar que una URL, es la abreviación inglesa de Uniform Resource Locator, estándar de definición para la localización de recursos en la red, para facilitar la lectura se utilizará el término de siglas inglesas.

A continuación, a modo recopilatorio, en el cuadro 3.1 se especifica el caso de uso con el que se inicia la actividad, centrándose en los actores del usuario y del departamento de marketing.

Nombre
Búsqueda de alquiler de coches
Descripción
Un usuario interesado en alquilar un coche para realizar un viaje comienza a hacer búsquedas en Internet con el objeto de encontrar una oferta que se ajuste a sus necesidades y presupuesto.
Actores
Los actores que intervienen son el usuario que realiza las búsquedas y el departamento de marketing.
Precondición
El usuario debe estar interesado en alquilar un coche por internet.
Flujo normal
El usuario hace búsquedas y accede al primer enlace que es de un anuncio de *DoYouSpain*.
Flujo alternativo
El usuario hace búsquedas y tras consultar varios enlaces de distintas compañias de alquiler acaba accediendo a *DoYouSpain* para realizar una reserva.
Post-condición
El equipo de marketing debe personalizar los anuncios de tal forma que aumenten las probabilidades de que el usuario acabe accediendo a sus anuncios, si no es en la búsqueda actual, para las futuras.

Cuadro 3.1: Especificación caso de uso: Búsqueda de alquiler de coches.

Dado el mismo escenario pero enfocado a otro actor, este caso de uso tiene otra especificación. El actor sobre el que radica la acción es el generador, encargado de recibir una petición de una URL, la cual debe interpretar y devolver el resultado oportuno. A continuación su especificación en detalle en el cuadro 3.2.

Nombre
Búsqueda de alquiler de coches
Descripción
Un usuario interesado en alquilar un coche para realizar un viaje comienza a realizar búsquedas en Internet y accede a una dirección web insertada en un anuncio de *DoYouSpain*.
Actores
Los actores que intervienen son el usuario que accede al anuncio y el generador que recibe la petición.
Precondición
El usuario debe acceder al enlace de un anuncio de *DoYouSpain*
Flujo normal
El generador recibe la petición, interpreta la dirección web, la valida, carga la información pertinente a dicha petición y devuelve la información con su diseño correspondiente al usuario.
Flujo alternativo
El generador recibe la petición, interpreta la dirección web, no la valida, notifica al usuario del error.
Post-condición
Tanto el generador haya recibido una dirección web válida como inválida el usuario debe estar informado en todo momento de la evolución de su petición, sin salir del marco de la página web.

Cuadro 3.2: Especificación caso de uso: Búsqueda de alquiler de coches.

3.2. Especificación de requisitos

Tras especificar los casos de uso del sistema desarrollado, se obtiene un serie de requisitos que deben ser cumplidos a fin de alcanzar de forma exitosa el buen desarrollo de la aplicación. Estos requisitos resultantes se pueden clasificar en tres tipos: requisitos de datos, requisitos de tecnología y requisitos de interfaces.

3.2.1. Requisitos de datos

Inicialmente, se debe conocer las variantes de construcción de las direcciones web, así como los condicionantes de cada una de ellas. Todo usuario de la red, cuando realiza búsquedas, deja un rastro.

Con la utilización de las herramientas adecuadas, ésta información puede ser recopilada e interpretada. El departamento de marketing se encarga de captar e interpretar esta información. Como puede observarse en la figura 3.4, herramientas tales como Google Adwords, Search Console, Google Analytics, entre otras, son utilizadas para analizar el tráfico de datos que se genera cuando se realizan búsquedas.

Figura 3.4: Obtención datos

Con el análisis intensivo de estos datos se han concluido aquellas construcciones que dan lugar a una mejor interpretación y visibilidad por el usuario. Estos datos no son estáticos, lo cual hace que la variabilidad de construcción sea creciente. Tal y como puede observarse en el cuadro 3.3, estos datos obtenidos sirven de fuente de datos para la consulta sobre una dirección web.

Requisitos de datos	
Código	RD01
Nombre	Consulta sobre una dirección web
Versión	1.0
Autores	Lara Gómez
Fuentes	Gesmarket S.L.
Requisitos asociados	-
Datos específicos	Parametros que forman la dirección web
Ocurrencias	1 – 1.000.000
Importancia	Alta
Comentarios	

Cuadro 3.3: Requisitos de datos para petición de una dirección web.

Una vez se introduce una dirección web, esta debe ser interpretada por el generador. Una vez interpretada se requiere consultar los datos que deben ser devueltos al usuario. A continuación, en el cuadro 3.4, se definen dichos requisitos.

Requisitos de datos	
Código	RD02
Nombre	Interpretación de la dirección web
Versión	1.0
Autores	Lara Gómez
Fuentes	Gesmarket S.L
Requisitos asociados	-
Datos específicos	Tipo destino, meta-etiquetas, imágenes, formato formulario, formato pie de cabeza, texto, módulos específicos, destinos, datos ficheros, promociones
Ocurrencias	1 – 1.000.000
Importancia	Muy alta
Comentarios	Para construir la interfaz que se devuelve al usuario, se debe previamente consultar toda la información asociada a los parámetros de la dirección, tales como destino, en el caso de que lleve, textos personalizados, imágenes,...etc. Todo va personalizado, por lo que siempre se han de consultar dichos datos.

Cuadro 3.4: Requisitos de datos de consulta de una dirección web

3.2.2. Requisitos de tecnología

El marco de tecnologías en el que debe integrarse el programa a desarrollar se describe a continuación.

Servidor:

- El programa debe estar desarrollado en C#.
- Microsoft SQL Server como gestor de base de datos.
- Para facilitar la manipulación de los datos contenidos en Microsoft SQL Server se ha utilizado el ORM Microsoft Entity Framework Core 2.1. ORM, del ingles Object Relational Mapping, es una técnica de mapeado de los datos almacenados en una base de datos relacional, en una estructura lógica de entidades,

con el objetivo de simplificar y acelerar el acceso y manipulación de los datos, sin la utilización del lenguaje SQL (Structured Query Language) propio de la base de datos.

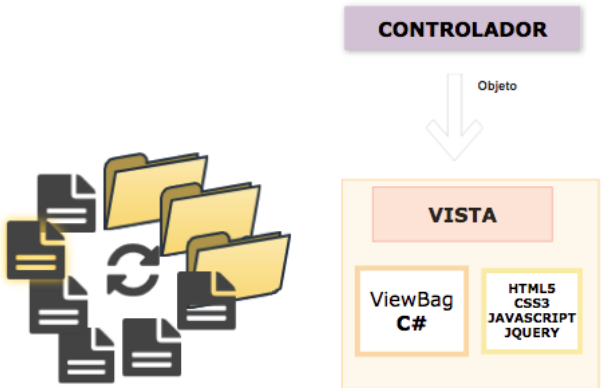
Cliente:

- Las interfaces deben definirse en ficheros de tipo .CSHTML, ya que permiten modelar el comportamiento de la vista realizando consultas en C# sobre objetos dentro del marcado HTML, dando lugar a una creación de vistas mas personalizada y por consiguiente, a la reducción del número de vistas a definir.
- El lenguaje de definición debe ser HTML5, CSS , JQUERY y JAVASCRIPT.
- Las interfaces deben ser *responsive*.

Estos requisitos han sido la base para establecer la arquitectura de Do, mas adelante se procederá a la descripción de dicha arquitectura.

3.2.3. Requisitos de interfaces

No se requiere la creación de ninguna interfaz nueva, solo se debe modificar la forma de construcción de las mismas.
Si bien con el actual generador, el comportamiento de cada vista está descrito en distintos ficheros .ASP que definen vistas parciales, ahora se pretende que cada bloque de la web tenga asociado un único fichero de tipo CSHTML. Cuando se invoca esta vista, se le pasa el objeto al cual hace referencia. De esta forma, a través el ViewBag (parámetro que almacena toda la información de esa instancia del objeto), se puede acceder a la información que requiera y así calcular la vista en base a los valores que tenga el objeto. Tanto el cálculo, como la definición de la vista para ese bloque de la página web, quedan definidos en un único archivo. En la figura 3.5 se puede ver de forma gráfica la diferencia entre ambos procesos.



(a) Generador actual (b) Do

Figura 3.5: Comparativa entre las formas de obtención de las vistas

3.2.4. Diseño de la arquitectura

La selección de la arquitectura sobre la cual se diseña una aplicación repercutirá en la eficacia, eficiencia y escalabilidad de dicha aplicación. Un claro ejemplo es lo ocurrido con el generador actual, en cuanto a la eficiencia en la generación no tiene réplica alguna pero si en cuanto a la escalabilidad. A continuación se explicará la arquitectura utilizada para la implementación del programa, así como la arquitectura del entorno software utilizado durante el desarrollo.

Arquitectura de implementación de la aplicación

Como anteriormente se ha explicado, el actual generador esta programado sobre ASP clásico. En un total de alrededor de 200 ficheros se describe la lógica y el diseño de las interfaces, además de albergar todos los datos que componen la página web.
Esto ha servido de precedente a la hora de determinar la arquitectura de la nueva versión del generador, que se ha basado en el principio de separación de intereses. Este principio consiste en separar y especializar toda la programación en capas, de tal forma que cada capa se encargue de procesar los datos recibidos y comunicar los resultados obtenidos. Todas las capas deben estar comunicadas de tal forma que genere un flujo de información que dé como resultado el objetivo de la aplicación, la generación de la página web solicitada.
La arquitectura seleccionada sigue un patrón de diseño de arquitectura de software am-

pliamente utilizado, se denomina Modelo-Vista-Controlador. Tal y como muestra la figura 3.6 está formada por tres capas.

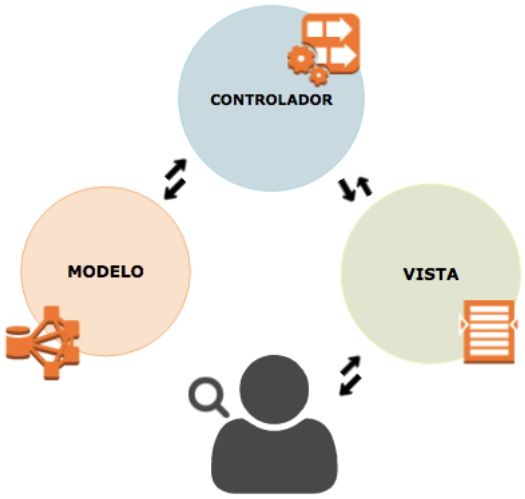


Figura 3.6: Patrón diseño: Modelo-Vista-Controlador

El usuario interacciona con la vista, que muestra los datos enviados por el controlador que a su vez, recibe los datos del modelo, capa encargada de captar los datos de la fuente donde se almacenen, base de datos, ficheros...etc.

Arquitectura del servidor

La arquitectura del servidor es la utilizada durante la fase de desarrollo. Al ser un proyecto tan grande y con varias fases por realizar a posteriori, no se llegó a implantar en producción. La figura 3.7 muestra la arquitectura utilizada.

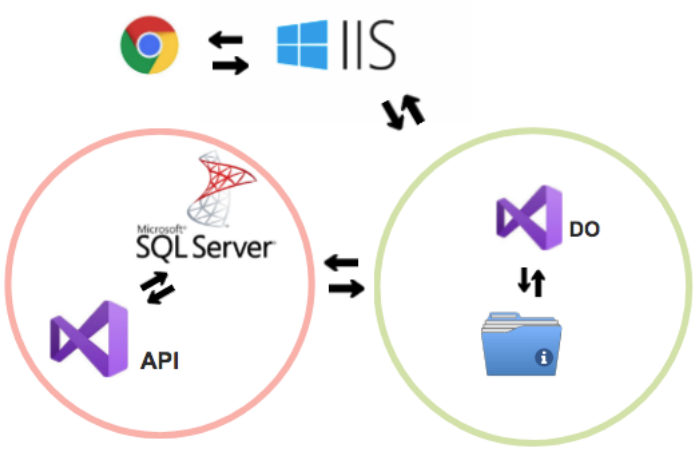


Figura 3.7: Arquitectura de trabajo del servidor

Inicialmente se ha utilizado el navegador Google Chrome para enviar las peticiones al servidor. Se ha elegido esta navegador por la gran variedad de herramientas que ofrece para depurar errores que se puedan generar al visualizar las interfaces. Como emulador del servidor se ha utilizado el IIS(Internet Information Services) pudiendo así realizar peticiones a través del navegador y obtener una respuesta inmediata. En este servidor, se ha configurado un sitio web al cual se han agregado las aplicaciones necesarias.
Primero la aplicación que proporciona el acceso a los datos, este acceso se ha encapsulado a través de una API con arquitectura REST. La segunda aplicación agregada es Do, donde se ha implementado el proyecto. También se ha agregado un directorio web que apuntaba a la carpeta donde se almacenan todas las imágenes de las interfaces. Para que ambas aplicaciones puedan comunicarse entre ellas y trabajar de forma simultánea deben estar configuradas en puertos lógicos distintos, de esta forma se garantizan ambos canales de comunicación.
El funcionamiento y características de ambas aplicaciones se explicarán en detalle en el capitulo 4.

3.2.5. Diseño de la interfaz

Tal y como se explicaba en los requisitos de interfaces, no se han tenido que diseñar interfaces nuevas, pero si se han tenido que recopilar y unificar las vistas. Si anteriormente las interfaces eran un conjunto de vistas parciales, ahora cada parte de la página web tiene asociada una vista. En esa vista se establecen y calculan todas las posibilidades de variación de diseño que pueda sufrir ese bloque de la página web. Las posibilidades varían según los parámetros enviados en la dirección web.

A continuación se listarán los distintos bloques en los que se ha divido la web. El nombre de los bloques está en inglés ya que es la nomenclatura original utilizada en el desarrollo.

Header

El *header*, figura 3.8, es común a todas las variantes de direcciones web, en cuestiones de diseño y texto.

Figura 3.8: Vista bloque *header* de la página web DoYouSpain

Form

Existen cuatro tipos de presentaciones del formulario. Todos ellos tienen en común que todos los textos van personalizados según el destino asociado. El formulario de la figura 3.9 va asignado a aquellos destinos categorizados como destinos populares. El fondo del formulario es una foto panorámica del propio destino, acompañado por una serie de reseñas sobre la empresa.

Figura 3.9: Vista bloque *form* tipo 1 de la página web DoYouSpain

El formulario de la figura 3.10 va asignado a aquellos destinos no categorizados como destinos populares, el fondo del formulario es blanco y a su lado aparece Claire, cabeza representante de atención al cliente de la empresa.

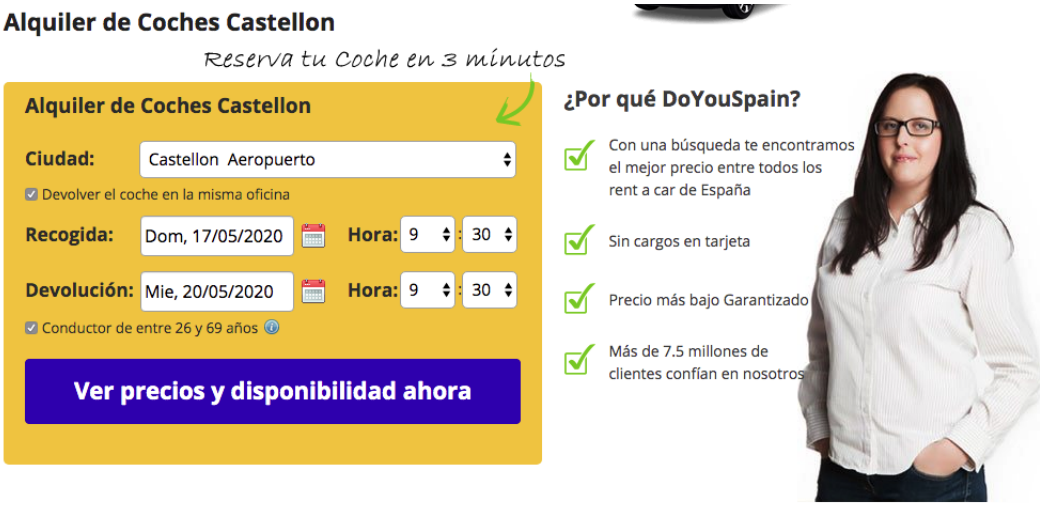


Figura 3.10: Vista bloque *form* tipo 2 de la página web DoYouSpain

El formulario de la figura 3.11 va asignado a los destinos populares que tienen asociados promociones.

Figura 3.11: Vista bloque *form* tipo 3 de la página web DoYouSpain

El formulario de la figura 3.12 va asociado a las páginas de error.

Figura 3.12: Vista*form* tipo 4 de la página web DoYouSpain

Rents

La figura 3.13 muestra un ejemplo de la vista del bloque *rents*. Este bloque está destinado a mostrar información sobre algunas empresas de alquiler con las que podrá alquilar un vehículo. La forma mas visual y rápida de reconocer una compañía es a través de su logo.

Figura 3.13: Vista bloque *rents* de la página web DoYouSpain

FaqLists

Las *faqlist* son un conjunto de frases que aportan información sobre los servicios que encontrará en la página web para el destino en el cual ha realizado la búsqueda. La figura 3.14 muestra un ejemplo de su vista.

Figura 3.14: Vista bloque *faqlist* de la página web DoYouSpain

Footer

El *footer*, tal y como muestra la figura 3.15, aporta información a través de enlaces que dan soporte a la atención e información del usuario, enlaces sobre otros destinos disponibles en lo que pudiese estar interesado, así como la posibilidad de cambiar el idioma en el que se muestra la página web. Por último, información legal acerca de la empresa.

Figura 3.15: Vista bloque *footer* de la página web DoYouSpain

Capítulo 4

Implementación

En este capítulo se detalla el proceso de implementación del nuevo generador Do para las *landings* de tipo dinámicas.

Como se ha citado en capítulos anteriores, se ha hecho uso de patrones de diseño [20]. Los patrones de diseño son técnicas utilizadas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño, para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

En este caso se han utilizado dos patrones de diseño, uno para la encapsulación de la persistencia de datos y otro para definir la arquitectura del software.

4.1. Encapsulamiento de datos

El origen de los datos a consultar es una fuente externa a la aplicación, parte de los datos se almacenan en una base de datos relacional. Inicialmente todas las peticiones de consulta de datos se realizaban directamente, a través de la clase DataContext. Esta clase perteneciente al espacio de nombres de System.Data.Linq se encuentra en el conjunto de librerías que ofrece .NET. Ha sido la primera opción por su sencillo uso y su fácil instalación, ya que estaba disponible a través del gestor de librerías, también conocido

como nuGet [21], de Visual Studio.

Según ha avanzado el desarrollo se ha ido incrementando el número de consultas así como el mapeado de los resultado a los objetos pertinentes, dando lugar a un código cada vez mas tedioso de depurar. Es por ello que se ha decidido encapsular el acceso a los datos externalizandose a otra aplicación web.

Esta aplicación web ha sido creada como un servicio web REST. Un servicio web REST, también denominado API REST, permite la interacción de máquina a máquina a través de una red. Utiliza el protocolo HTTP sin estado y requiere una URI para enviar la petición sobre el recurso solicitado. La figura 4.1 muestra el proceso de comunicación para la obtención de datos.

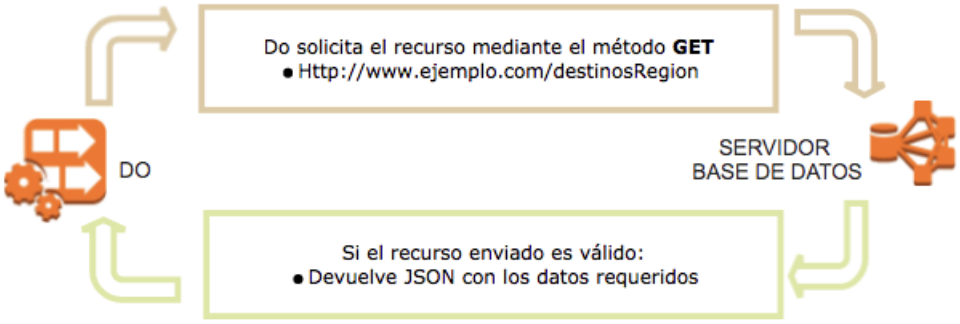


Figura 4.1: Comunicacion de recursos a través del servicio web REST

En la API están definidas un conjunto de funciones que dan servicio a todas las consultas posibles sobre la base datos. Do realiza peticiones a dicha API, la cual devuelve el resultado en formato JSON. Será Do, una vez recibida la respuesta, quien se encargue de mapear el resultado con el objeto correspondiente definido. Todas las peticiones que se realizan a través de esta API dedicada, son de tipo GET, ya que para la funcionalidad que se esta desarrollando no necesitamos modificar ni generar ningún registro nuevo en la base de datos, solo necesitan ser extraídos.

Todo este proceso de encapsulamiento para la extracción de datos utiliza el patrón de diseño DAO (Data Access Object). El DAO encapsula el acceso a la/s fuente/s de datos (DataSources), de manera que los detalles son transparentes para el cliente, siendo un cliente cualquier objeto de la aplicación que necesite la persistencia de datos. Aunque en este caso solo se haya utilizado la operación GET, da soporte a las típicas operaciones CRUD (CREATE, READ, UPDATE, DELETE).

4.2. Modelo-Vista-Controlador

Como se comentaba en el capitulo 3 en relación a la arquitectura de sistema, el patrón de diseño utilizado ha sido el Modelo-Vista-Controlador. Con este tipo de arquitectura se consigue la separación de los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos y separados.
El proyecto se ha realizado, al igual que con el servicio web anteriormente descrito, con el programa Microsoft Visual Studio. Este entorno de desarrollo nos brinda un conjunto variado de plantillas con proyectos predefinidos. Se ha seleccionadi una plantilla para el tipo aplicación web MVC con el *framework* .NET, entendiendo el concepto de *framework* como un entorno de trabajo con una estructura conceptual y tecnológica que da asistencia en el desarrollo de aplicaciones. Esta asistencia consiste en el soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Esta plantilla ya establece un directorio de carpetas predefinido para este tipo de proyecto, el cual no se ha modificado y ha servido como punto de partida.

4.2.1. Modelo

Esta capa es la encargada de asegurar la persistencia de datos, se encarga de mapear todos los datos devueltos por la API, por lo cual es necesario tener clases definidas para cada unos de los tipos de objetos que puedan existir. Se generará una instancia del objeto que el controlador necesite para cada petición de una dirección web.
En esta capa también se han definido todas las estructuras que componen las direcciones web, junto con todos aquellos atributos que puede tener cada estructura. La variabilidad que existe en la combinación de las distintas estructuras junto con los atributos predefinidos, sitio web e idioma son atendidos por la capa controlador.

4.2.2. Controlador

El controlador, capa intermedia, es la encargada de interpretar las peticiones del usuario y gestionar las operaciones necesarias entre el resto de capas, para proporcionar al usuario una respuesta. Es la única capa que se comunica con todas las capas restantes.
Si se inspecciona la estructura de las *landings* dinámicas se vería que la operatividad que

se le ofrece al usuario, una vez dentro de la propia página, es mínima. Las operaciones que se realizan a través del controlador, para esta parte de la web, son únicamente dos: el acceso a la propia página y construcción de la misma, y la funcionalidad del formulario. Ambas funcionalidades operan en dos controladores distintos, pero que se instancian a la par por cada petición. Esta homogeneidad hace que no sea necesario el uso de patrones de enrutamiento. Como particularidad a destacar de ambos controladores es la utilización de técnicas que aprovechan la caché, la asincronicidad y el multihilo.
A continuación se explicará varios procesos en los que se han utilizado dichas técnicas.

Memoria caché

La utilización de la memoria caché ha sido necesaria para mejorar la eficiencia de la aplicación. Cada petición recibida por el controlador requiere la consulta de datos a través de la API, para evitar la consulta repetida de datos y el incremento del tiempo de carga, se ha procedido a almacenar todos los datos consultados en memoria caché. En el caso de que los datos buscados no se encuentran en memoria, se procede a consultar a la fuente de datos. Ccon esta técnica se ha reducido drásticamente el número de peticiones a la base de datos.
Existen diferentes memorias caché disponibles donde almacenar dichos datos. Para este proyecto se ha utilizado la del navegador del cliente. El uso de esta memoria se ha configurado con la creación de una clase encargada de la carga de todos los datos, con la particularidad del atributo [OutputCache]. Adicionalmente este atributo tiene una serie de parámetros configurables para determinar el tiempo de expiración de la memoria así como la ubicación de almacenamiento.

Asincronicidad

La encapsulación del acceso a los datos a través de una API supone la dependencia de la red de datos sobre la que se envía la petición y recibe la respuesta. Dependiendo de las características de aplicación las consultas pueden realizarse de forma síncrona o asíncrona. La diferencia entre ambos radica en que si bien con métodos síncronos la ejecución espera a la devolución del dato, con el método asíncrono la ejecución y devolución del dato se realiza de forma paralela.
Para Do, el acceso a datos de forma asíncrona combinado con la memoria caché, anteriormente explicado, ha sido la técnica utiliza por sus buenos resultados de tiempo de carga.

Multihilo

Todas las construcciones válidas de direcciones web, alrededor de dos millones por *site* se almacenan en una base datos no relacional dedicada. La validación de la dirección web es una operación crítica cuyo tiempo de respuesta debe ser lo mas reducido posible. La utilización de la base de datos no relacional de MongoDB [22] con el uso de hebras para el reparto de carga de procesamiento de las peticiones, supuso la reducción del tiempo de carga de 2 segundos a 200 milisegundos.

4.2.3. Vistas

Cada bloque de la web varia en función de los parámetros de la dirección web. Es por ello que las vistas están dividas en bloques. Las vistas son archivos CSHTML que usan el lenguaje de programación C# con el marcado de Razor. Cuando se instancia una vista, junto con la instanciación se envía el objeto al cual hace referencia la vista, con todos los datos del objeto previamente consultados y calculados en el controlador.

El tipo de datos que se utiliza para enviar los datos del controlador a la vista es del tipo específico *ViewBag*. Su persistencia se mantiene sólo durante la petición actual, y una vez termina, el dato es borrado. Tal y como muestra la figura 4.2, inicialmente se realiza la conversión de tipos, el marcado Razor es el código comprendido entre los caracteres @{ y }. Se realiza un *casting*(conversión entre variables) del objeto recogido por el ViewBag a la clase especifica del objeto.

```
@{
    var layout = (DoWeb.Web.Layout)ViewBag.Layout;
    var idioma = (DoWeb.Web.Idioma)ViewBag.Idioma;
    var footer = (DoWeb.Web.Footer)Page.footer;
    var linkIdiomas = (Dictionary<DoWeb.Web.Idioma, string>)Page.linkIdiomas;
    var site = (DoWeb.Web.Site)ViewBag.Site;
    var nombreSite = DoWeb.Web.WebClass.sites[site].nombre.ToLower() + ".com";

    var articleClass = (layout.HasFlag(DoWeb.Web.Layout.Desktop) ? "genBlock" : "genBlock genBlock__collapsible collapsed");
    var displaySection = (layout.HasFlag(DoWeb.Web.Layout.Desktop) ? "" : "display: none;"); ;
    var footerNuevoMovil = false; // No se aplica nuevo footer en movil hasta confirmacion
}
```

Figura 4.2: Acceso a datos del objeto a través Viewbag

Una vez instanciado el objeto especifico, se puede acceder a los datos almacenados en él y así personalizar la vista con los datos del objeto. En la figura 4.3 se muestra un ejemplo.

```
<!-- footer 1 -->

<div class="container-fluid footer footer1">
    <div class="container">
        <div class="row">
            <div class="col-xs-3">
                <article class="@articleClass">
                    <header>
                        <span class="genBlock__title">@Html.Raw(footer.footer1_1_title)</span>
                    </header>
                    <section style="@displaySection">
                        <div class="genBlock__content">
                            <ul class="clearlist">
                                <li><a href="@footer.link_customerSupport" rel="noopener" target="_blank">@Html.Raw(footer.customerSupport)</a></li>

                                <li><a href="@footer.link_opinionsUser" rel="noopener" target="_blank">@Html.Raw(footer.opinionsUsers)</a></li>
                                <li><a href="@footer.link_contactUs" rel="noopener" target="_blank">@Html.Raw(footer.contactUs)</a></li>
                            </ul>
                        </div>
                    </section>
                </article>
            </div>
```

Figura 4.3: Uso marcado Razor

Las ventajas de esta forma de generar las vistas son muchas:

- Reducción de duplicidad de código por la reutilización.
- Facilidad de búsqueda de las vistas relacionadas cuando se trabaja en una característica.
- Es más fácil probar los elementos de la interfaz de usuario de la aplicación, ya que las vistas son unidades independientes.
- Reducción de la casuística de generación de errores.

El resultado obtenido de la implementación de lo anteriormente citado ha sido prometedor pero el proceso de desarrollo ha sido costoso, ya que la única parte que ha sido posible reutilizar era referente a diseño e interactividad de la parte cliente, es decir, archivos HTML, CSS, JAVASCRIPT y JQUERY. El objetivo era replicar el servicio de tal forma que el usuario no percibiese ningún cambio, pero a nivel *back-end* la funcionalidad y lógica debía especializarse y unificarse. El objetivo ha sido alcanzado con éxito, reduciéndose tiempos de carga de la página web, mejor control de errores y lo mas importante, reducción del tiempo a invertir en la mejora, modificación de cualquier *landing* existente así como la creación de nuevas.

El desarrollo de este proyecto ha continuado y continúa evolucionando, en la actualidad se está implantando en producción.

Capítulo 5

Verificación y validación

Esta capitulo muestra las técnicas utilizadas para testear el funcionamiento de Do. Estas técnicas se han aplicado a lo largo de todo el desarrollo como medida de apoyo para la comprobación del correcto progreso. En la etapa final la verificación y validación ha sido más exhaustiva, ya que el objetivo era la búsqueda de errores y su minimización.

5.1. Test generar direcciones web

El desarrollo del proyecto comenzó con la creación del conjunto de direcciones web válidas. Una vez definidas las estructuras, su construcción consistía en una función que combina todas las posibilidades. Para comprobar que las combinaciones eran correctas, se creó una consulta al servicio web que nos devolvía un JSON con todas las direcciones web que cumplían los condicionantes de la petición.
En la figura 5.1 se puede ver la función que, dado un idioma y un *site*, devuelve una lista de tipo *string* con todas las direcciones web que cumplen las condiciones. Como se puede observar, tal y como se ha comentado en el capitulo 4, las peticiones a la base de datos son asíncronas.

```
[HttpGet]
[Route("test/pathswebs/dinamicas/{site}/{idioma}")]
[ResponseType(typeof(List<string>))]
0 referencias | Lara, Hace 22 días | 2 autores, 5 cambios
public async Task<HttpResponseMessage> PathsWebDinEN(string site, string idioma)
{
    try
    {
        Idioma idiom = (DoWeb.Web.Idioma)Enum.Parse(typeof(DoWeb.Web.Idioma), idioma);
        Site sit = (DoWeb.Web.Site)Enum.Parse(typeof(DoWeb.Web.Site), site);

        if (!Start.IsDesarrollo() && !Start.IsGesmarket())
            throw new NotImplementedException();

        var obj = await PathsWebStatic.EntityPathWeb.GetPathsWeb(TipoWeb.Dinamica, idiom, sit, true);

        return Request.CreateResponse(System.Net.HttpStatusCode.OK, obj.Select(i => i.Key));
    }
    catch (Exception ex)
    {
        return Request.CreateErrorResponse(System.Net.HttpStatusCode.BadRequest, "Error:" + ex.Message + ex.StackTrace);
    }
}
```

Figura 5.1: Petición HTTP para listado URLS

De esta forma la consulta a la API, se realiza a través del programa Postman que formatea el resultado. Así, la detección de direcciones web incorrectas es muy fácil.

5.2. Test peticiones HTTP

Para comprobar que el conjunto de direcciones web generadas funcionase correctamente, se ha creado una función que por cada dirección web recibida realizara una petición *request HTTP*, definiéndose HTTP como protocolo de transferencia de hipertexto.
En las figuras 5.2 y 5.3 se puede ver el funcionamiento. Se realiza una petición de tipo GET y dependiendo del valor devuelvo por el protocolo HTTP se crea una respuesta.

```
[HttpGet]
[Route("test/responsedo")]
[ResponseType(typeof(string))]
0 referencias | Lara, Hace 6 días | 3 autores, 8 cambios
public async Task<HttpResponseMessage> ResponseDo(string urlPage)
{
    try
    {
        string redirection = CacheDo.GetRedirection(urlPage.ToString());
        if (redirection != null)
            return Request.CreateResponse(System.Net.HttpStatusCode.OK, "301;" + redirection);
        string host = new Uri(urlPage).Host;
        urlPage = urlPage.Replace("https://", "");
        urlPage = urlPage.Replace("http://", "");
```

Figura 5.2: Petición HTTP para realizar peticiones

En vez de devolver sólo el código del protocolo, se ha considerado mejor devolver un tipo *string* formado por el código junto la dirección web a la que hace referencia. Con esto se consigue un mejor análisis del resultado, al saber que dirección web es la afectada en caso de recibir un 404.

```
            if (await CacheDo.GetExistUrlPathsWeb(urlPage.ToString(), site))
            {
                return Request.CreateResponse(System.Net.HttpStatusCode.OK, "200;" + urlPage);
            }
            else
            {
                return Request.CreateResponse(System.Net.HttpStatusCode.OK, "404;" + urlPage);
            }
        }
        catch (Exception ex)
        {
            return Request.CreateErrorResponse(System.Net.HttpStatusCode.BadRequest, "Error:" + ex.Message + ex.StackTrace);
        }

    }
```

Figura 5.3: Resultado petición HTTP

Las peticiones se envían de forma masiva. Para facilitar el análisis del resultado, se ha añadido la funcionalidad de visualizar las respuestas a un programa ya existente en la empresa desarrollado por mis compañeros. En la figura 5.4 se puede ver parte de la respuesta obtenida para un test de 500 direcciones web para el idioma inglés.

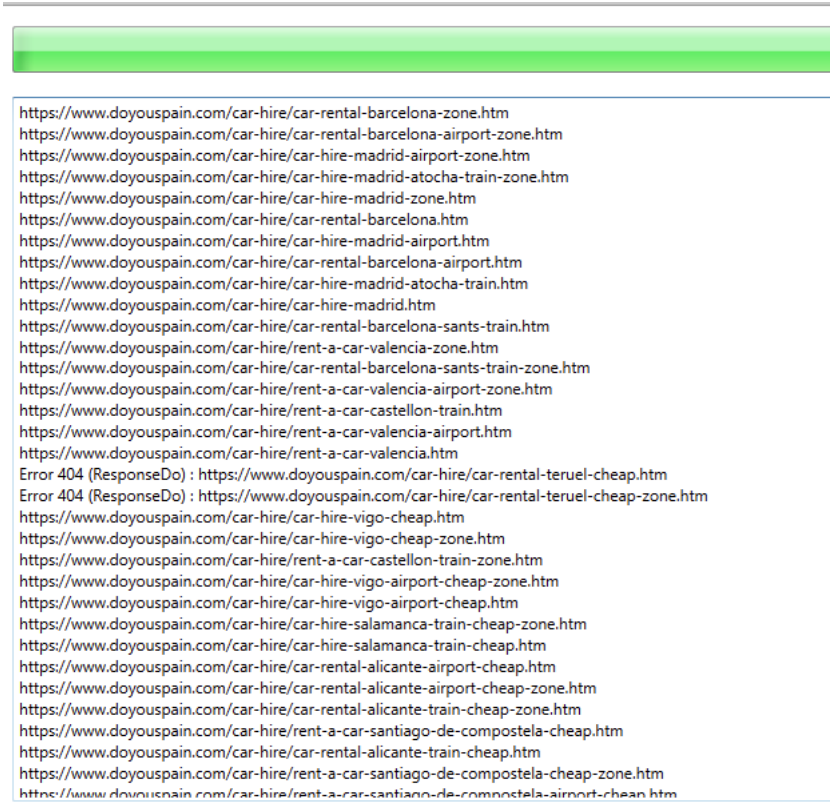


Figura 5.4: Interfaz con el conjunto de resultados de varias peticiones

Evaluación de interfaces

Para la evaluación de las interfaces, un compañero de la empresa ha realizado varios test heurísticos. El objetivo final del test ha sido garantizar que las interfaces se hubiesen reproducido con, exactamente, el mismo comportamiento a las que debían reemplazar. Se han realizado numerosos cambios en las interfaces hasta conseguir el apto en todas las preguntas. En los cuadros del 5.1 al 5.11 se puede ver los test realizados.

Generales	**Apto**
¿Cuáles son los objetivos del sitio web? ¿Son concretos y bien definidos?	✓
¿Los contenidos y servicios que ofrece se corresponden con esos objetivos?	✓
¿Tiene una URL correcta, clara y fácil de recordar? ¿Y las URL de sus páginas internas? ¿Son claras y permanentes?	✓
¿Muestra de forma precisa y completa qué contenidos o servicios ofrece realmente el sitio web?	✓
¿La estructura general del sitio web está orientada al usuario?	✓
¿El look & feel general se corresponde con los objetivos, características, contenidos y servicios del sitio web?	✓
¿Es coherente el diseño general del sitio web?	✓
¿Es reconocible el diseño general del sitio web?	✓
¿El sitio web se actualiza periódicamente? ¿Indica cuándo se actualiza?	✓

Cuadro 5.1: Test evaluación aspectos generales

Identidad e información	**Apto**
¿Se muestra claramente la identidad de la empresa-sitio a través de todas las páginas?	✓
El logotipo, ¿es significativo, identificable y suficientemente visible?	✓
El eslogan o tagline, ¿expresa realmente qué es la empresa y qué servicios ofrece?	✓
¿Se ofrece algún enlace con información sobre la empresa, sitio web, 'webmaster',...?	✓
¿Se proporciona información sobre la protección de datos de carácter personal de los clientes o los derechos de autor de los contenidos del sitio web?	✓

Cuadro 5.2: Test evaluación identidad e información

Lenguaje y redacción	**Apto**
¿El sitio web habla el mismo lenguaje que sus usuarios?	✓
¿Emplea un lenguaje claro y conciso?	✓
¿Es amigable, familiar y cercano?	✓
¿1 párrafo = 1 idea?	✓

Cuadro 5.3: Test evaluación lenguaje y redacción

Rotulado	**Apto**
Los rótulos, ¿son significativos?	✓
¿Usa rótulos estándar?	✓
¿Usa un único sistema de organización, bien definido y claro?	✓
¿Utiliza un sistema de rotulado controlado y preciso?	✓
El título de las páginas, ¿es correcto? ¿ha sido planificado?	✓

Cuadro 5.4: Test evaluación sobre rotulado

***Layout* de la página**	**Apto**
¿Se aprovechan las zonas de alta jerarquía informativa de la página para contenidos de mayor relevancia?	✓
¿Se ha evitado la sobrecarga informativa?	✓
¿Es una interfaz limpia, sin ruido visual?	✓
¿Existen zonas en "blanco.^entre los objetos informativos de la página para poder descansar la vista?	✓
¿Se hace un uso correcto del espacio visual de la página?	✓
¿Se utiliza correctamente la jerarquía visual para expresar las relaciones del tipo "parte de.^entre los elementos de la página?	✓
¿Se ha controlado la longitud de página?	✓

Cuadro 5.5: Test evaluación sobre *layout*

Estructura y navegación	**Apto**
La estructura de organización y navegación, ¿es la más adecuada?	✓
En el caso de estructura jerárquica, ¿mantiene un equilibrio entre profundidad y anchura?	✓
En el caso de ser puramente hipertextual, ¿están todos los nodos comunicados?	✓
¿Los enlaces son fácilmente reconocibles como tales? ¿Su caracterización indica su estado (visitados, activos,...)?	✓
En menús de navegación, ¿se ha controlado el número de elementos y de términos por elemento para no producir sobrecarga memorística?	✓
¿Es predecible la respuesta del sistema antes de hacer clic sobre el enlace?	✓
¿Se ha controlado que no haya enlaces que no llevan a ningún sitio?	✓
¿Existen elementos de navegación que orienten al usuario acerca de dónde está y cómo deshacer su navegación?	✓
Las imágenes enlace, ¿se reconocen como clicables? ¿incluyen un atributo 'title' describiendo la página de destino?	✓
¿Se ha evitado la redundancia de enlaces?	✓
¿Se ha controlado que no haya páginas huérfanas?	✓

Cuadro 5.6: Test evaluación sobre estructura y navegación

Búsqueda	**Apto**
¿Se encuentra fácilmente accesible?	✓
¿Es fácilmente reconocible como tal?	✓
¿Permite la búsqueda avanzada?	✓
¿Muestra los resultados de la búsqueda de forma comprensible para el usuario?	✓
¿La caja de texto es lo suficientemente ancha?	✓
¿Asiste al usuario en caso de no poder ofrecer resultados para una consulta dada?	✓

Cuadro 5.7: Test evaluación realización búsquedas

Elementos multimedia	**Apto**
¿Las fotografías están bien recortadas? ¿son comprensibles? ¿se ha cuidado su resolución?	✓
¿Las metáforas visuales son reconocibles y comprensibles por cualquier usuario?	✓
¿El uso de imágenes o animaciones proporciona algún tipo de valor añadido?	✓
¿Se ha evitado el uso de animaciones cíclicas?	✓

Cuadro 5.8: Test evaluación elementos multimedia

Ayuda	**Apto**
Si posee una sección de ayuda, ¿Es verdaderamente necesaria?	✓
El enlace a la sección de ayuda, ¿está colocado en una zona visible?	✓
¿Se ofrece ayuda contextual en tareas complejas?	✓

Cuadro 5.9: Test evaluación ayuda ofrecida

Accesibilidad	**Apto**
¿El tamaño de fuente se ha definido de forma relativa, o por lo menos, la fuente es lo suficientemente grande como para no dificultar la legibilidad del texto?	✓
¿El tipo de fuente, efectos tipogrficos, ancho de línea y alineación empleados facilitan la lectura?	✓
¿Existe un alto contraste entre el color de fuente y el fondo?	✓
¿Incluyen las imágenes atributos 'alt' que describan su contenido?	✓
¿Es compatible el sitio web con los diferentes navegadores? ¿se visualiza correctamente con diferentes resoluciones de pantalla?	✓
¿Puede el usuario disfrutar de todos los contenidos del sitio web sin necesidad de tener que descargar e instalar plugins adicionales?	✓
¿Se ha controlado el peso de la página?	✓
¿Se puede imprimir la página sin problemas?	✓

Cuadro 5.10: Test evaluación accesibilidad

Control y retroalimentación	**Apto**
¿Tiene el usuario todo el control sobre el interfaz?	✓
¿Se informa constantemente al usuario acerca de lo que está pasando?	✓
¿Se informa al usuario de lo que ha pasado?	✓
Cuando se produce un error, ¿se informa de forma clara y no alarmista al usuario de lo ocurrido y de cómo solucionar el problema?	✓
¿Posee el usuario libertad para actuar?	✓
¿Se ha controlado el tiempo de respuesta?	✓

Cuadro 5.11: Test evaluación control y retroalimentación

De los aspectos evaluados, aquellos que mayor tiempo de depuración supusieron han sido los relativos a las direcciones web, textos a mostrar y tiempo de respuesta.
El problema con la construcción de las direcciones web es su elevada casuística. Al estar sujeta a diferentes condicionantes relativos al idioma, el *site*, el destino... se ha tenido que depurar bastante el método de construcción para generar, única y exclusivamente, las direcciones web válidas.
El problema concerniente a los textos radica en la gran cantidad de textos a gestionar. Con el anterior generador todos los textos se leen de distintos archivos ASP, por lo que se deduce que no se almacenaban en ninguna base de datos. Hubo que recopilar todos los textos, referenciarlos con una clave lo más homogénea posible, ya que los textos podían o no estar restringidos a diferentes *sites*, e introducirlos en la base de datos. Dato importante que tuve que tener en cuenta a la hora de comprobar que los textos eran los correctos tras un cambio que se hubiese realizado, era el purgado de la caché. Con purgado me refiero a la acción de eliminar todos los datos almacenados en la memoria del navegador, si no se realizaba, los cambios no se hacían efectivos.
Para igualar los tiempos de carga, que finalmente han sido mejorados, se tuvo que hacer varios cambios en la implementación. Reducción y unificación de las llamadas a la API encargada de la persistencia de datos y migración de la estructura que almacena los objetos referentes a las direcciones web a una base de datos no relacional.

Capítulo 6

Resultados y conclusiones

La estancia en prácticas, así como el desarrollo de este proyecto, ha supuesto un experiencia única de la cual he extraído una serie de conclusiones y resultados que a continuación detallaré.

6.1. Resultados

El resultado de la estancia en prácticas ha sido muy positivo ya que se ha conseguido alcanzar, en su totalidad, el objetivo inicial del proyecto. Los resultados parciales que se han ido consiguiendo durante el desarrollo han ido mejorando gracias a los cambios de implementación y tecnología aplicados. El desarrollo de este generador ha sido el inicio de la migración de toda la infraestructura de la web DoYouSpain, por lo cual existen muchas mejoras y ampliaciones que realizar.

Las mejoras inmediatas posibles a realizar, y que he continuado realizando tras finalizar las prácticas, ha sido extender la creación de las *landings* dinámicas al resto de *sites*, DoYouItaly y Carjet. También el resto de *landings* que forman parte de la página web se irán incluyendo en la construcción.

Una posible mejora a realizar sería la unificación de todas las estructuras de diseño de las *landings*, tanto dinámicas como las estáticas para todos los *sites*. De tal forma que los cambios únicamente radiquen en aquellos derivados de textos o diseños específicos.

Actualmente, se esta migrando la construcción de las *landings* dinámicas en producción

con Do, de forma progresiva. Se ha iniciado con todos los idiomas de móvil y con el idioma Portugués para *desktop*. Realizarlo de forma progresiva por idioma es debido a que, si bien como se explicaba al inicio de la memoria, este tipo de páginas están destinadas para campañas de posicionamiento SEM, recibir un error 404 en un enlace insertado en un anuncio es algo a evitar a toda costa, ya que se esta invirtiendo dinero en un enlace que no da el servicio deseado.

La migración por idioma aumenta la posibilidad de detección de errores y rápida solución además de evitar que se reproduzcan para el resto de idiomas, ya que han sido solventados.

6.2. Conclusiones

La realización de este proyecto ha supuesto un reto a varios niveles. A nivel formativo, el desarrollo de un proyecto con perspectivas de implantarse en una producción real es un escenario totalmente distinto a lo vivido durante los estudios del grado, en el que todos los proyectos se desarrollan en un entorno controlado. Esto aporta mayor motivación a la par que mayor precaución en su desarrollo. También ha supuesto una gran inversión de horas en formación aunque sin el aprendizaje adquirido durante el grado hubiese sido mas compleja su consecución. Finalmente los objetivos a alcanzar han sido superados en su totalidad y el proyecto sigue creciendo y mejorándose.

A nivel personal, la experiencia ha sido muy gratificante no sólo por el proyecto desarrollado sino por el equipo humano que forma la empresa. Sin la ayuda y apoyo de mis compañeros hubiera sido imposible alcanzar dicho resultado. Mi confianza ha salido fortalecida y con ganas de seguir aprendiendo.

Bibliografía

[1] Wikipedia, la enciclopedia libre. ASP clásico.
Recuperado de https://es.wikipedia.org/wiki/Active_Server_Pages
[Consultado 10 Mayo 2020]

[2] Wikipedia, la enciclopedia libre. Modelo–vista–controlador.
Recuperado de https://es.wikipedia.org/wiki/Modelo %E2 %80 %93vista %E2 %80 %93controlador
[Consultado 16 Mayo 2020]

[3] API.
Recuperado de https://juanda.gitbooks.io/webapps/content/api/arquitectura-api-rest.html
[Consultado 16 Mayo 2020]

[4] Scrum.
Recuperado de https://proyectosagiles.org/que-es-scrum/
[Consultado 16 Mayo 2020]

[5] Microsoft. Documentación de ASP.NET.
Recuperado de https://docs.microsoft.com/es-es/aspnet/core/?view=aspnetcore-3.1
[Consultado 1 Abril 2020]

[6] Wikipedia,la enciclopedia libre. Git gestor versiones
Recuperado de https://es.wikipedia.org/wiki/Git
Consultado 10 Marzo 2020]

[7] Tortoise.
Recuperado de https://tortoisegit.org/about/
[Consultado 10 Marzo 2020]

[8] Microsoft. MSQL management
Recuperado de https://es.wikipedia.org/wiki/Microsoft_SQL_Server
[Consultado 10 Marzo 2020]

[9] Microsoft. I.I.S
Recuperado de https://es.wikipedia.org/wiki/Internet_Information_Services
[Consultado 10 Marzo 2020]

[10] Microsoft. Visor de eventos
Recuperado de https://docs.microsoft.com/eses/windows/security/threatprotection/microsoft-defenderatp/eventerrorcodes
[Consultado 10 Marzo 2020]

[11] Postman.
Recuperado de https://www.postman.com/product/api-client/
[Consultado 15 Abril 2020]

[12] Wikiepedia, la enciclopedia libre. Google
Recuperado de https://es.wikipedia.org/wiki/Google_Chrome
[Consultado 10 Marzo 2020]

[13] Microsoft. C#
Recuperado de https://docs.microsoft.com/es-es/dotnet/csharp/
[Consultado 10 Marzo 2020]

[14] Wikipedia, la enciclopedia libre. Javascript.
Recuperado de https://es.wikipedia.org/wiki/JavaScript
[Consultado 15 Abril 2020]

[15] Wikipedia, la enciclopedia libre. HTML5.
Recuperado de https://es.wikipedia.org/wiki/HTML5
[Consultado 15 Abril 2020]

[16] Wikipedia, la enciclopedia libre. CSS.
Recuperado de https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascadanewline [Consultado 15 Abril 2020]

[17] Wikipedia, la enciclopedia libre. SQL.
Recuperado de https://es.wikipedia.org/wiki/SQL
[Consultado 15 Abril 2020]

[18] Wikipedia, la enciclopedia libre. Visual Studio
Recuperado de https://es.wikipedia.org/wiki/Microsoft_Visual_Studio
[Consultado 10 Marzo 2020]

[19] Instituto Nacional de Estadística.
Recuperado de https://www.ine.es/jaxiT3/Tabla.htm?t=3693 [Consultado 10 Marzo 2020]
[Consultado 10 Marzo 2020]

[20] Wikipedia, la enciclopedia libre. Patrón de diseño.
Recuperado de https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o
[Consultado 10 Mayo 2020]

[21] Microsoft. Instalación y uso de un paquete NuGet en Visual Studio.
Recuperado de https://docs.microsoft.com/es-es/nuget/quickstart/install-and-use-a-package-in-visual-studio
[Consultado 16 Mayo 2020]

[23] MongoDB
https://www.mongodb.com/es
[Consultado 16 Junio 2020]